An Asynchronous Distributed Algorithm for Real-Time Travel-Time Prediction in Vehicular Traffic Networks

Karl E. Wunderlich

University of Michigan
Ann Arbor

The University of Michigan
IVHS
Intelligent Vehicle-Highway Systems
College of Engineering • (313) 764-4332
Professor Kan Chen, 4112 EECS, Ann Arbor, MI 48109-2122, and
University of Michigan Transportation Research Institute • (313) 936-1066
Robert D. Ervin, 2901 Baxter Road, Ann Arbor, MI 48109-2150

IVHS Technical Report: #92-11

An Asynchronous Distributed Algorithm for Real-Time Travel-Time Prediction in Vehicular Traffic Networks

Karl E. Wunderlich

University of Michigan
Ann Arbor

IVHS Technical Report: #92-11

AN ASYNCHRONOUS DISTRIBUTED ALGORITHM FOR REAL-TIME TRAVEL-TIME PREDICTION IN VEHICULAR TRAFFIC NETWORKS

by K.E. Wunderlich
May 27, 1992

Abstract

Several current methods for link travel-time prediction in traffic networks use an iterative method to circumvent computational complexities arising from the interdependence of forecasted link travel-times and vehicles receiving user-optimal route guidance. Often, a simulation is used in conjunction with a dynamic fastest-path calculation. For large-scale networks, however, the time required to find fixed points associated with dynamic equilibrium is often longer than the solution horizon. For this reason, these current serial methods do not converge quickly enough to provide real-time route guidance information, even on today's most powerful computers.

One possible solution is to perform much of the simulation and dynamic fastest-path calculation in parallel, using asynchronous distributed processing. An alternative parallel algorithm is presented for the serial SAVaNT iterative routing/assignment method for anticipatory route guidance. The parallel algorithm is shown to slightly out-perform the serial approach in the worst case, and a speedup of near 7.5 in the average case, without requiring any specialized parallel architecture. Finally, an extension of the real-time problem as an infinite-horizon problem is included, as is a brief discussion of the implementation of parallel concepts within a discrete-event traffic simulation.

1: Introduction

Accurate, dynamic, link-travel-time prediction is a task central to the Intelligent Vehicle/Highway System (IVHS) goal of improving real-time traffic network performance, both in terms of reduced total system travel-time and experienced travel-times of individual drivers. Traditional traffic assignment models are static in nature; that is, they produce link loadings and resultant travel-times based on an average case over periods as long as a day. These methods fail, however, when faced with shorter solution horizons. For traditional transportation infrastructure planning, these static models were sufficient. However, the IVHS concept requires accurate link-time prediction in a dynamic fashion, updated on a minute-by-minute basis.

In general, the dynamic link-time prediction problem has as its input a set of time-varying demands from a set of origins to corresponding destinations, network topology and capacity, and some natural relation between network congestion and travel-time. Outputs include projected time-dependent link flows and corresponding travel-times for all links in the network for some solution horizon (usually corresponding to a rush hour period). When the number of vehicles receiving route guidance information is small, the effect these vehicles have on network congestion conditions are negligible. Hence, we may route them according to dynamic shortest routes using a modified version of Dijkstra's algorithm[1]. However, when sizeable percentages of vehicles moving on the network receive route guidance, even if we route according to some forecasted shortest routes, the guided vehicles may cause unanticipated congestion on the network and thus render the route guidance suboptimal.

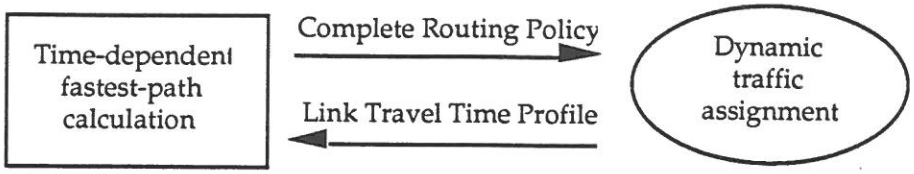

Figure 1 Feedback between link travel time forecasts and route guidance

This fundamental interdependence of predicted travel-times and route guidance is illustrated graphically as a feedback loop in figure 1. Several current approaches attempt to circumvent this feedback by breaking the tasks into two independent processing entities, one of which calculates time-dependent fastest-paths and one that performs dynamic traffic assignment. Most often, a simulation is used for the traffic assignment phase. Examples include Jayakrishnan and Mahmassani[2] and Rillett and Van Aerde[3]. Kaufman, Smith and Wunderlich[4] define anticipatory route guidance as paths calculated from a set of predicted link travel-times, which when disseminated to drivers cause the same set of predicted link travel-times to be realized by vehicles in the network. Thus, the paths distributed as route guidance were based on a correct assumption about congestion conditions on the network.

Combining a time-dependent fastest-path calculation program with a traffic simulation in an iterative manner, we can identify a complete set of paths (called a routing policy) as anticipatory if in any two iterations, the link travel-time profiles are sufficiently similar. Kaufman et al[4] demonstrate that this approach can indeed identify such anticipatory routing policies, which when disseminated, result not only in conditions similar to the ones forecasted, but also in enhanced traffic network performance.

This iterative routing/assignment method[4], also known as the Simulation of Anticipatory Vehicle Network Traffic (SAVaNT) method, has previously been used only in a serial fashion. For the small test networks

cited in their paper, this method generates anticipatory routing policies relatively quickly. However, for larger networks, this current serial method may take several hours to identify anticipatory routing policies. In fact, the time to fixed point (the point at which we identify the anticipatory policy) for realistic regional traffic networks in the SAVaNT loop may be too long to enable the dissemination of the identified policy as real-time route-guidance, even when implemented on today's fastest computers.

One possible solution is to perform much of the simulation and dynamic fastest-path calculation in parallel, using asynchronous distributed processing. Section 2 is an algorithmic statement of the serial iterative method. Section 3 presents an efficient time-dependent fastest-path algorithm, which can be implemented in parallel with the traffic simulation. Section 4 provides an analysis of the interaction of the dynamic router and the simulation in parallel, including speedup and efficiency analysis for worst and average cases. Finally, in Section 5, an extension of the real-time problem as an infinite-horizon problem is posed and discussed in terms of its theoretical difficulties. A brief discussion is also included on possible implementation of parallel concepts within a discrete-event traffic simulation.

2 Serial Approach and Notation

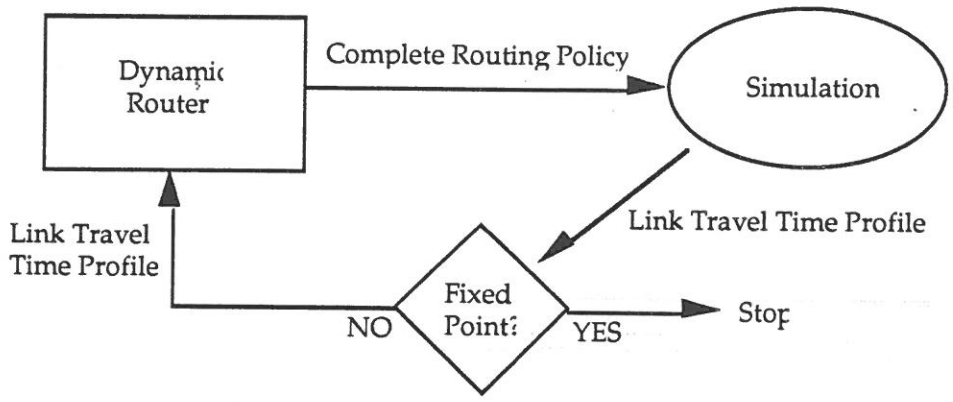

Figure 2 The SAVaNT iterative method.

Figure 2 illustrates how SAVaNT uses an iterative method to solve for anticipatory routing policies. When two consecutive link travel-time profiles differ by less than some small threshold value ε , SAVaNT halts; otherwise, the most recent profile is used as the forecast for the dynamic route, which performs time-dependent fastest-path calculations to produce a new routing policy.

In discrete time, a routing policy π is composed of time-dependent next link information. Consider $\pi_i^d(t)$ an instruction to take link $\ell=(i,j)$ if you depart node i in time period t with destination d. As in the previous section, we identify the complete routing policies generated in each iteration of SAVaNT with a subscript, e.g., π_1.

The simulation has a solution horizon of H discrete time periods. Typically, the duration of a single time period is about 60 seconds, although this may vary depending on the implementation. A further assumption in SAVaNT is that all vehicles can finish their trips by time period H. Therefore, after some time $H' < H$ no vehicles are allowed to enter the network, which clears by H. From results using small test networks, $H = 2H'$

appears sufficient for this condition. Network capacity and time-dependent O-D demand are given entities.

A link travel-time profile is composed of dynamic link travel-times, $c_\ell(t)$ experienced by a vehicle that begins traversing link ℓ during time period t such that $c_\ell(t) \geq 0 \; \forall \, \ell, t$. A complete profile C has values for all links in the network and for all $t = 1, 2, 3, \cdots, H$. We identify each travel-time profile by iteration with a subscript, e.g., C_1.

To generate an initial point for SAVaNT, we generate a "free flow" travel-time profile C_0 where travel-time for each link in all time periods is set to a constant. This constant represents the time required to traverse the link if no other cars were on the link, i.e., $c_\ell(t) = c_\ell^0 \; \forall t$ and ignores the effect of signalization.

The iterative method can be described with the following algorithm:

Step 0.	INITIALIZATION
0.1	Compute the initial routing policy, $\pi_0 = \boldsymbol{R}(C_0)$ This generates $\pi_i^d(t) \; \forall \, i, d; t = 1, 2, 3, \cdots, H$.
0.2	Set the iteration counter, $n = 1$.
Step 1.	SIMULATION.
1.1	Set $c_\ell(1) = c_\ell^0 \; \forall \, \ell$.
1.2	Launch simulation n. *(DO until $t = H$)*
1.2a	• Vehicles departing from any node in the network take a next link according to $\pi_i^d(t)$
1.2b	• Exponential Smoothing of Travel Time Profile When any vehicle finishes traversing a link during time period t, set $c_\ell(t) = \alpha c_\ell(t) + (1 - \alpha)\tau_\ell$ where $0 \leq \alpha \leq 1$ and τ_ℓ = travel-time experienced by departing vehicle.
1.2c	• At the end of time period t, write $c_\ell(t)$ to disk. Set $c_\ell(t+1) = c_\ell(t) \; \forall \, \ell$.
1.3	Collect the $c_\ell(t)$ values written to disk as C_n.
Step 2.	TRAVEL TIME PROFILE COMPARISON. If $\lvert C_n - C_{n-1} \rvert < \varepsilon$, then STOP.

Otherwise, set the iteration counter, $n = n+1$. CONTINUE.

Step 3. DYNAMIC ROUTER.

3.1 Compute $\pi_n = \boldsymbol{R}(C_n)$ using dynamic programming, i.e.:

$$f_i^d(t) = \min_{\ell=(i,j)\in SCS(i)} \left\{ c_\ell(t) + f_j^d\left(t + c_\ell(t)\right) \right\}$$

where $f_i^d(t)$ = duration (in time periods) of the shortest path for vehicles departing node i during the time period t with destination d.

$f_d^d(t) \equiv 0 \; \forall \; d, t, n.$

$SCS(i)$ = the set of all links that begin at i.

Note that the $\pi_i^d(t)$ values for π_n are generated in Step 3.1.

3.2 GOTO Step 1.

Note that in step 1.2b, we use exponential smoothing to handle the time series of reported link travel-times, although other methods could be employed, e.g., simple averaging.

3: Parallel Time-Dependent Fastest Path Calculation

Consider the situation where we have multiple processors, some of which can perform the task of the dynamic router, and others that can perform the task of the simulation. Instead of waiting for complete travel-time profiles to be sent to the dynamic router, travel-times for each time period t are sent directly to the appropriate processor when the simulation reaches time $t+1$. In turn, when a dynamic router processor is able to prepare a routing policy for some given time period, that partial policy is made available immediately to the next waiting simulation processor. The parallel concept is illustrated in figure 3.

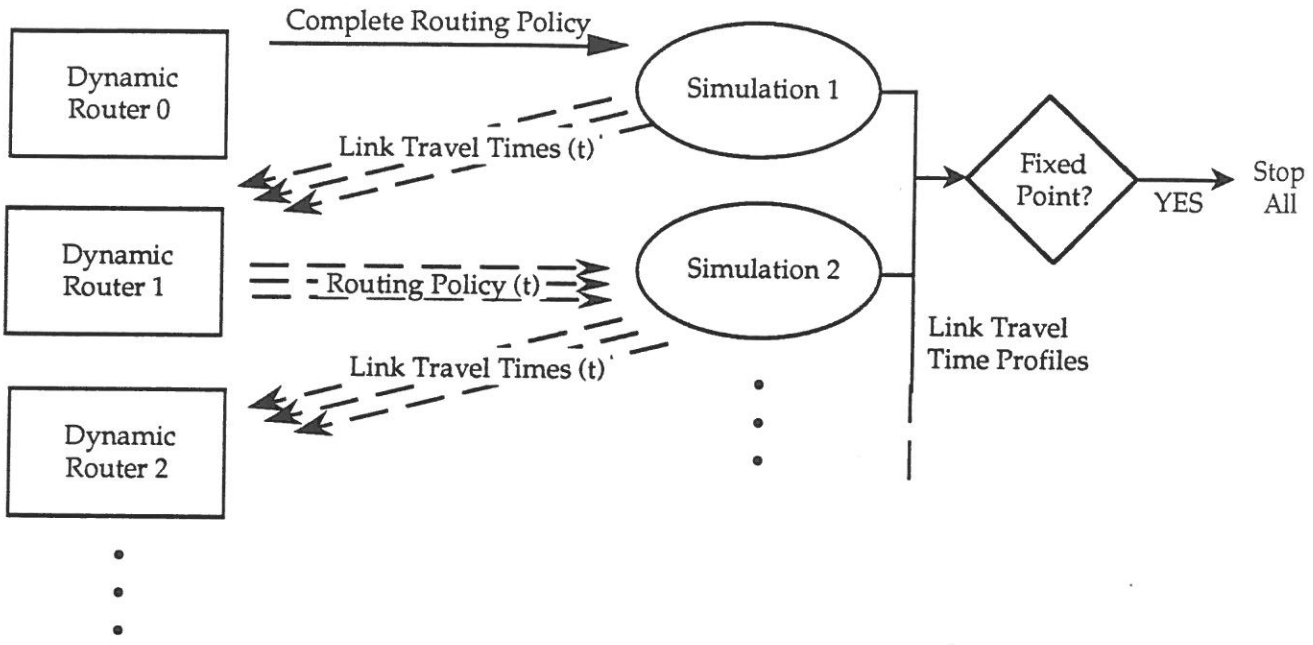

Fig ure 3. Asynchronous, distributed approach

Note that from the serial algorithm, if we wish to advance a single time period in simulation, we need only to have next-link information for vehicles that depart nodes during that time period. All other pertinent data regarding network capacity, topology and congestion effects are already known. Consider Dynamic Router processor #1, DR1. As described in Step 1.2b, at times $t = 1, 2, 3, \cdots, H$, the exponentially smoothed travel-times for all the links on the network, $(c_\ell(t) \forall \ell)$ are being communicated to it from Simulation processor #1, SIM1. For the real-time anticipatory route guidance problem, our goal is to provide fastest paths for the k pairs of origins and destinations which have guided vehicles on them at time $t = 0$ (for general t, call this set of O-D pairs $K(t)$). We are interested in calculating $\pi_i^d(0) \forall (i,d) \in K(0)$ quickly so that we may begin SIM2. In general, one may ask the question: how long must we let SIM(n) run before we can identify $f_{O^{(k)}}^{D^{(k)}}(t)$ the fastest travel-time for one of the k O-D pairs contained in $K(t)$? When $f_{O^{(k)}}^{D^{(k)}}(t)$ is identified, we may also find $\pi_{O(k)}^{D(k)}(t)$.

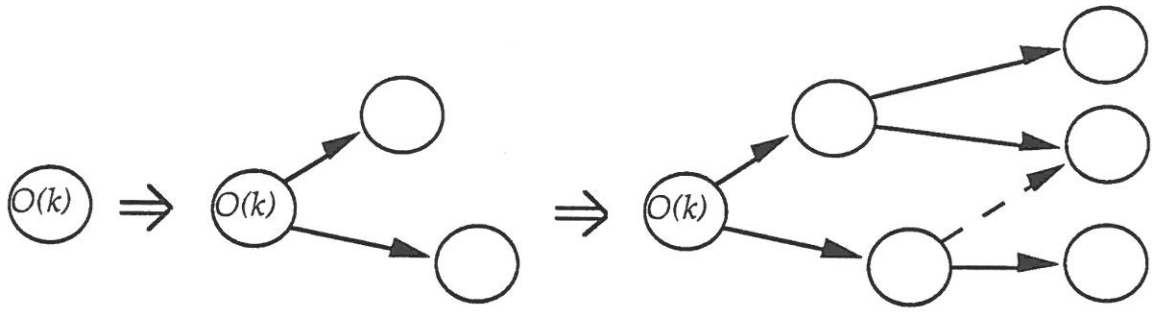

Figure 4. The set of all nodes reachable from $O(k)$ grows as more time slices pass by.

Within an iteration, let $F_{O^{(k)}}^{(t,t')}$ be the set of all nodes reachable from $O^{(k)}$ if a vehicle left at time t and traveled until $t', t' > t$. Clearly $F_{O^{(k)}}^{(t,t)} = \{O^{(k)}\}$. Define $F_{O(k)}^{(t)}$ as the union all $F_{O^{(k)}}^{(t,t')}$ identified in real-time. Note that as $t \to H, \left|F_{O(k)}^{(t)}\right| \uparrow$, as shown in figure 4.

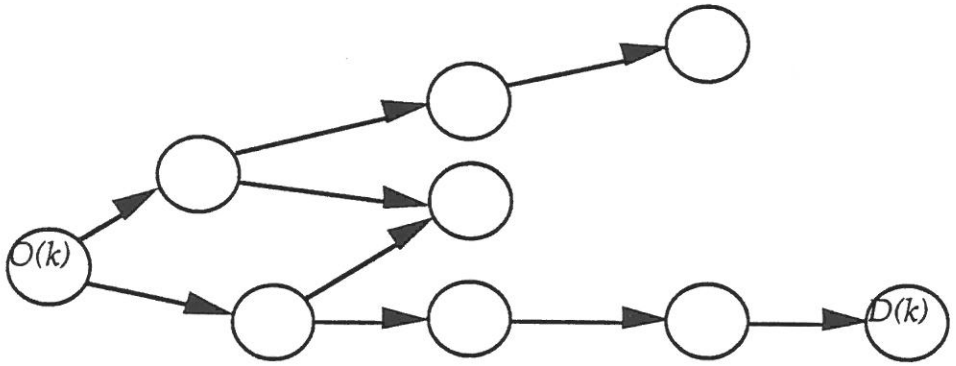

Figure 5. When $D(k)$ joins the set of reachable nodes then we may calculate a fastest path.

At some time $t'' < H, D^{(k)}$ joins $F_{O(k)}^{(t)}$, as shown in figure 5.

<u>Claim</u>: At t'' we have enough information to find $f_{O^{(k)}}^{D^{(k)}}(t)$.

<u>Pf:</u> Assume our claim is false.

Then there exists $i' \notin F_{O(k)}^{(t)}$ and a link $\ell' = \left(i', D^{(k)}\right)$ such that $f_{O^{(k)}}^{i'}(t) + c_{\ell'}\left(t + f_{O^{(k)}}^{i'}(t)\right) < t''$

Since $c_\ell(t) \geq 0 \; \forall \; \ell, t$ and $i' \notin F_{O(k)}^{(t)}$ implies $f_{O^{(k)}}^{i'}(t) > t''$

this is obviously a contradiction.

Since we have identified a subnetwork that is sufficient for finding a fastest path, one could naively return to $O^{(k)}$ and perform dynamic

programming on this subnetwork to find a fastest path from $O^{(k)}$ to $D^{(k)}$. In fact, at each time period t, the dynamic router could be maintaining a fastest-path tree to all nodes and at t'' have only to perform a trace back along this tree to identify the fastest path. This trace takes less time to compute than solving for a shortest path in the entire subnetwork.

The general parallel dynamic fastest-path algorithm is then:

Step 0: Initialization.

0.1 Set $t' = t$; $f^{O^{(k)}}_{O^{(k)}}(t) = 0$; $\boldsymbol{F}^{(t,t')}_{O^{(k)}} = \{O^{(k)}\}$.

0.2 Set $f^{i}_{O^{(k)}}(t) = \infty \ \forall\, i \neq O^{(k)}$.

Step 1: RECEIVE $c_\ell(t) \ \forall\, \ell$

1.1 $t' = t' + 1$

1.2 IF $f^{i}_{O^{(k)}}(t) + c_\ell\left(t + f^{i}_{O^{(k)}}(t)\right) < t'$ for any $i \in \boldsymbol{F}^{(t)}_{O^{(k)}}$, $\ell = (i,j)$ then

1.2a Set $\boldsymbol{F}^{(t)}_{O^{(k)}} = \boldsymbol{F}^{(t)}_{O^{(k)}} \cup \{j\}$.

1.2b Set $f^{j}_{O^{(k)}}(t) = \min\left\{f^{j}_{O^{(k)}}(t)\,,\, f^{i}_{O^{(k)}}(t) + c_\ell\left(t + f^{i}_{O^{(k)}}(t)\right)\right\}$.

ENDIF

Step 2: TRANSMIT

2.1 IF $D^{(k)} \in \boldsymbol{F}^{(t)}_{O^{(k)}} \ \forall\ \left(O^{(k)}, D^{(k)}\right) \in \boldsymbol{K}^{(t)}$ then

2.1a TRANSMIT $\pi^{d}_{i}(t) \ \forall\, i$

ENDIF

2.2 GOTO Step 1 and wait for next message from simulation.

4: Asynchronous Distributed Fixed-Point Calculation

It appears we will be able to do some of both the dynamic routing and simulation work in parallel. But exactly how much parallelism can be achieved is not clear. We can launch new simulations as soon as route guidance is provided for vehicles with demand at time $t = 0$. Say we can find fastest paths for all demands at $t = 0$ by the end of the fourth time period (See figure 6) DR1 can begin working at $t = 1$, and has only to do a simple trace at $t = 4$ before passing routes to SIM2. SIM2 is launched with its own local time

clock set to $t = 0$. Assume that the simulation runs in real time at some constant rate per time period; that is, the time to simulate a single time period is the same, independent of which time period is being simulated.

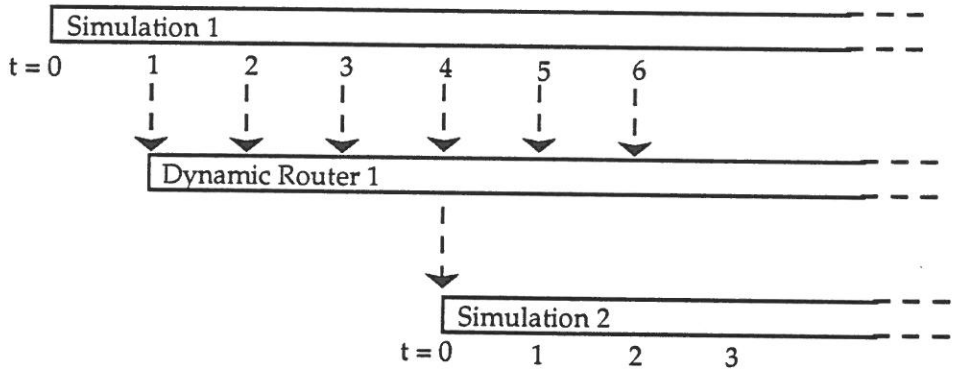

Fig ure 6. Parallel simulation launch with initial delay

The above example seems to indicate that we can do quite a bit of work in parallel--in this case, all of the fastest-path calculation as well as all of the simulation with the exception of first four time periods. However, this may not always be the case.

Define $\bar{f}(t)_n \equiv \max_{k \in K^{(t)}} \left\{ f_{O^{(k)}}^{D^{(k)}}(t) \right\}$,

the longest duration (in time periods) among fastest paths for vehicles departing from nodes at time t in iteration n. What happens when $\bar{f}(t+1)_n > \bar{f}(t)_n + 1$? Consider our previous example, with $\bar{f}(0)_n = 4$ and $\bar{f}(1)_n = 6$. As shown in figure 7, fastest-path routes cannot be prepared and readied for SIM2 at its local clock time of $t = 1$.

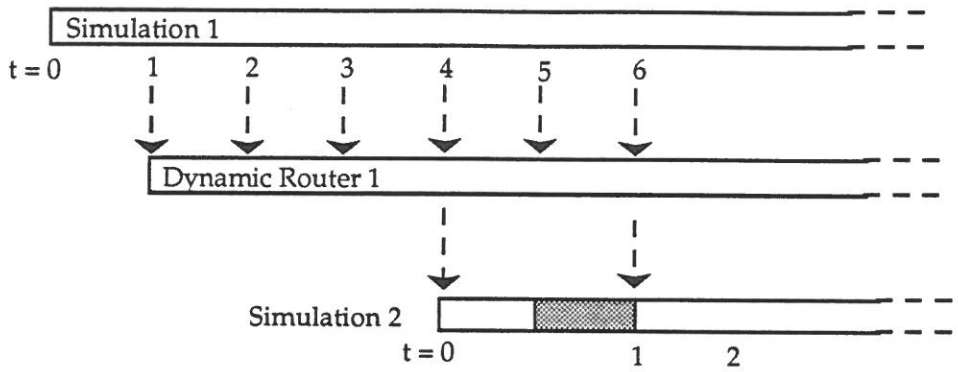

Fig ure 7. Wait-time condition: simulation 2 must halt and wait for routes at $t = 1$.

It could be, then, that simulations must halt even after they have been launched because of conditions in the previous iteration. Can we bound these wait times? First, define $W_n(t)$ as the number of time periods the simulation in iteration n has paused waiting for routing information to simulate vehicles at local time t. Second, define W_n as the total wait-time for simulation in iteration n. To illustrate how these entities can be calculated, consider the following example, in which our two simulations are now the $(n-1)$st and nth simulations (figure 8).

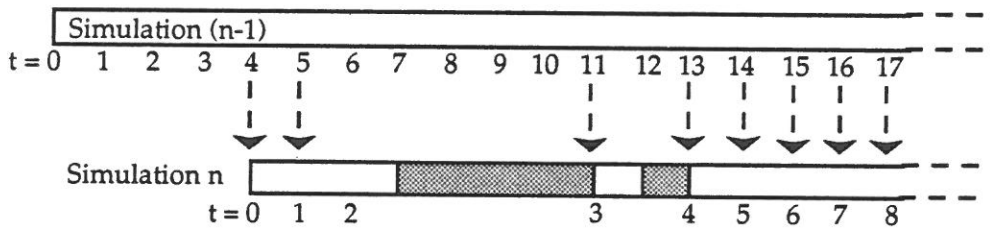

Figure 8. Wait-time example.

Say $\bar{f}(0)_{n-1} = 4$, $\bar{f}(3)_{n-1} = 8, \bar{f}(4)_{n-1} = 9$, $\bar{f}(8)_{n-1} = 9$, and $\bar{f}(t)_{n-1} = 1$ for $t = 1,2,5,6,7,9,10$. Our condition, $\bar{f}(t+1)_n > \bar{f}(t)_n + 1$, occurs at $t = 0,3,4,8$. But note that we don't incur any delay in SIM(n) for $t = 8$. Why? This is because the call for the $t = 8$ route guidance itself has been delayed by the long time needed to find paths at $t = 4$.

Define $R_n(t)$ as the earliest time on the (n-1)st clock that SIM(n) may begin work on its local time period t. We know that if there is no delay from previous work, we will begin work at time $\bar{f}(t)_{n-1}+t$. Otherwise, we will be delayed by the cumulative effect of previous waits. In general, then,

$$R_n(t)=\max_{0\le t'\le t}\left\{t'+\bar{f}(t')_{n-1}+(t-t')\right\}=\max_{0\le t'\le t}\left\{\bar{f}(t')_{n-1}+t\right\},$$

and $W_n(t)=R_n(t)-t$. Clearly, $W_n=\max_t\{W_n(t)\}=W_n(H-1)$.

Claim: $W_n\le H$.

Pf: Since all vehicles can finish travel by time H,

$\bar{f}(t)_{n-1}\le H-t$ for $t=1,2,3,\cdots,H$.

Assume our claim is false. Then $W_n>H$.

$\Rightarrow \max_t\{W_n(t)\}>H$

This implies that there exists some t^0 such that

$W_n(t^0)=R_n(t^0)-t^0>H$

$\Rightarrow \max_{0\le t'\le t^0}\{\bar{f}(t')_{n-1}+t^0\}-t^0>H$

we have $\bar{f}(t')_{n-1}\le H-t'$

$\Rightarrow H-t'+t^0-t^0>H$

$\Rightarrow H-t'>H$. But $t'\ge 0$. *Proof by contradiction.*

Therefore, in the worst case, we cannot do worse than serial performance. Remember that we have assumed that the amount of time required to simulate a time period is considered constant. Therefore, in the serial case, time-to-fixed-point can be described as:

$$N(bH+U+Z)$$

where N is the number of iterations to convergence; b the time to simulate a single time period; U the amount of time to find all time-dependent fastest-paths in the network; and Z is the time to compare two complete travel-time profiles.

In parallel, time-to-fixed-point is:

$$bH + \sum_{i=2}^{N} bW_i + Z$$

In the worst case, $W_i = H \; \forall \, i$, implies a time-to-fixed-point of $N(bH)+Z$. Note that $N(bH)+Z < N(bH+U+Z)$ and thus, the parallel method always finds fixed points faster than the serial method.

To compute an average case estimate, the test network from [13] was used in which extra lines of code had been added to detect condition $\bar{f}(t+1)_n > \bar{f}(t)_n + 1$. In 16 runs under varying intensity of network loadings, the condition never occurred except for the initial wait-to-launch. It seems that for most networks of any size, nearly every node has demand within a 60 second time period. Thus, we are comparing longest routes in the network over time, which tend to grow and decline in a gradual fashion. In the average case, then, $W_i = W_i(0)$ for $i = 2$ to N.

For large scale networks like the Troy network (500 nodes, 900 links, 26,000 vehicles), $W_i(0) \approx 0.1H$. Other typical values are as follows: $N = 25$ iterations; $H = 200$ minutes; $bH = 60$ minutes. Assume $Z = U = 0$. Under these conditions, average time-to-fixed-point for the serial method is 25[1 hour] = 25 hours. Average time-to-fixed-point for the parallel method is 60 minutes + 24 [6 minutes] = 204 minutes = 3.4 hours. Thus, the speedup rating for the parallel method is 7.35. It can be shown that for the above conditions, ten sets of router-simulator processors are required. Hence, the efficiency rating for the parallel method is .735.

5: Extensions and Conclusions

One simplifying assumption made in the iterative methods above is that there is some time horizon at which the network clears. In real-time

application on a real traffic network, of course, there is no time at which all vehicles can finish trips. One can then pose the link time prediction problem as an infinite-horizon problem. This would imply a simulation that ran for infinite time! Wunderlich[5] exhibits a brief example which demonstrates that simulating for horizons $H \leq \bar{f}(0)_n$ so-called "myopic" approach, can result in suboptimal routings and poor network performance. In practice, planners may choose some horizon deemed "long enough."

However, it may be possible to implement a tie-breaking algorithm, as in Ryan, Bean, and Smith[6], which can discover when a simulation has run for a sufficient horizon in order to provide accurate route guidance. Once such an algorithm can be shown to be applicable for the link travel-time prediction problem, then true real-time networks may be considered.

Another possible extension is to achieve further speedups in the iterative method by implementing distributed processing within the simulation processor. Clearly, several subroutines can be made to run in parallel, most notably synchronous distributed computation of link travel-times and signalization phase changes. More ambitious asynchronous methods include the time-warp distributed simulation approach of Jefferson[7]. However, the nearly continuous interaction between vehicles makes asynchronous approaches less attractive, as future scheduled events would be almost invariably have to be unsent. Romeijn and Smith[8] demonstrate the usefulness of geographic node aggregation in shortest path calculation and this may also be extended to include aggregation for simulation.

In conclusion, the asynchronous distributed method is appealing because it achieves speedup over the serial method and may be implemented without requiring a specialized parallel computer. The dynamic router and

simulator "processors" could be envisioned as a set of PCs on a LAN sharing information in real-time. Since the time to communicate between processors is considered to be much smaller than the processing time required between data transmissions, communication time is not considered above.

---

[1] Kaufman, D.E. and Smith, R.L. (1990) Minimum Travel time Paths in Dynamic Networks with Application to Intelligent Vehicle/Highway Systems. Technical Report 90-11, IVHS Program, The University of Michigan.

[2] Jayakrishnan R. and Mahmassani H. (1991) A Dynamic Modeling Framework of Real-Time Route Guidance Systems in General Urban Traffic Networks. Presented at the Transportation Technologies Applications Conference, Minneapolis, Minnesota.

[3] Rillet L. and Van Aerde M. (1991) Routing Based on Anticipated Travel Times. Presented at the Transportation Technologies Applications Conference, Minneapolis, Minnesota.

[4] Kaufman D. E., Smith R.L., and Wunderlich K.E. (1991) An Iterative Routing/Assignment Method for Anticipatory Real-Time Route Guidance. IEEE VNIS Conference Proceedings, Dearborm, Michigan. pp 693-700.

[5] Wunderlich, K.E. (1992) Notes on Real-Time Anticipatory Routing Posed as an Infinite Horizon Optimization Problem. University of Michigan (in progress).

[6] Ryan S.M., Bean J.C., and Smith R. L. (1991) A Tie-Breaking Rule for Discrete Infinite Horizon Optimization. To appear in Mathematical Programming.

[7] Jefferson, D. (1987) Distributed Simulation and the Time Warp Operating System. ACM Operating Systems Review.

[8] Romeijn, E. and Smith R.L. (1992) Parallel Algorithms and Aggregation for Solving Shortest Path Problems. University of Michigan IVHS Program (in progress).